VISUALIZING AND DISCOVERING BEHAVIOURAL WEAKNESSES IN DEEP REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

As deep reinforcement learning is being applied to more and more tasks, there is a growing need to better understand and probe the learned agents. Visualizing and understanding the decision making process can be very valuable to comprehend and identify problems in the learned behavior. However, this topic has been relatively under-explored in the reinforcement learning community. In this work we present a method for synthesizing states of interest for a trained agent. Such states could be situations (e.g. crashing or damaging a car) in which specific actions are necessary. Further, critical states in which a very high or a very low reward can be achieved (e.g. risky states) are often interesting to understand the situational awareness of the system. To this end, we learn a generative model over the state space of the environment and use its latent space to optimize a target function for the state of interest. In our experiments we show that this method can generate insightful visualizations for a variety of environments and reinforcement learning methods. We explore these issues in the standard Atari benchmark games as well as in an autonomous driving simulator. Based on the efficiency with which we have been able to identify significant decision scenarios with this technique, we believe this general approach could serve as an important tool for AI safety applications.

1 INTRODUCTION

Humans can naturally learn and perform well at a wide variety of tasks, driven by instinct and practice; more importantly, they are able to justify why they would take a certain action. Artificial agents should be equipped with the same capability, so that their decision making process is interpretable by researchers. Following the enormous success of deep learning in various domains, such as the application of convolutional neural networks (CNNs) to computer vision (LeCun et al., 1998; Krizhevsky et al., 2012; Long et al., 2015; Ren et al., 2015), a need for understanding and analyzing the trained models has arisen. Several such methods have been proposed and work well in this domain, for example for image classification (Simonyan et al., 2013; Zeiler and Fergus, 2014; Fong and Vedaldi, 2017), sequential models (Karpathy et al., 2016) or through attention (Xu et al., 2015).

Deep reinforcement learning (RL) agents also use CNNs to gain perception and learn policies directly from image sequences. However, little work has been so far done in analyzing RL networks. We found that directly applying common visualization techniques to RL agents often leads to poor results. In this paper, we present a novel technique to generate insightful visualizations for pre-trained agents.

Currently, the generalization capability of an agent is—in the best case—evaluated on a validation set of scenarios. However, this means that this validation set has to be carefully crafted to encompass as many potential failure cases as possible. As an example, consider the case of a self-driving agent, where it is near impossible to exhaustively model all interactions of the agent with other drivers, pedestrians, cyclists, weather conditions, even in simulation. Our goal is to extrapolate from the training scenes to novel states that induce a specified behavior in the agent.

In our work, we learn a generative model of the environment as an input to the agent. This allows us to probe the agent's behavior in novel states created by an optimization scheme to induce specific actions in the agent. For example we could optimize for states in which the agent sees the only option as being to slam on the brakes; or states in which the agent expects to score exceptionally low. Visualizing such states allows to observe the agent's interaction with the environment in critical scenarios to understand its shortcomings. Furthermore, it is possible to generate states based on an

objective function specified by the user. Lastly, our method does not affect and does not depend on the training of the agent and thus is applicable to a wide variety of reinforcement learning algorithms.

2 Related Work

We divide prior work into two parts. First we discuss the large body of visualization techniques developed primarily for image recognition, followed by related efforts in reinforcement learning.

2.1 Feature Visualization

In the field of computer vision, there is a growing body of literature on visualizing features and neuron activations of CNNs. As outlined in Grün et al. (2016), we differentiate between *saliency methods*, that highlight decision-relevant regions given an input image, methods that synthesize an image (pre-image) that fulfills a certain criterion, such as *activation maximization* (Erhan et al., 2009) or *input reconstruction*, and methods that are perturbation-based, i.e. they quantify how input modification affects the output of the model.

Saliency Methods Saliency methods typically use the gradient of a prediction or neuron at the input image to estimate importance of pixels. Following gradient magnitude heatmaps (Simonyan et al., 2013) and class activation mapping (Zhou et al., 2016), more sophisticated methods such as guided backpropagation (Springenberg et al., 2014; Mahendran and Vedaldi, 2016), excitation backpropagation (Zhang et al., 2016), GradCAM (Selvaraju et al., 2016) and GradCAM++ (Chattopadhay et al., 2018) have been developed. Zintgraf et al. (2017) distinguish between regions in favor and regions speaking against the current prediction. Sundararajan et al. (2017) distinguish between sensitivity and implementation invariance.

An interesting observation is that such methods seem to generate believable saliency maps even for networks with random weights (Adebayo et al., 2018). Kindermans et al. (2017b) show that saliency methods do not produce analytically correct explanations for linear models and further discuss reliability issues in Kindermans et al. (2017a).

Perturbation Methods Perturbation methods modify a given input to understand the importance of individual image regions. Zeiler and Fergus (2014) slide an occluding rectangle across the image and measure the change in the prediction, which results in a heatmap of importance for each occluded region. This technique is revisited by Fong and Vedaldi (2017) who introduce blurring/noise in the image, instead of a rectangular occluder, and iteratively find a minimal perturbation mask that reduces the classifier's score, while Dabkowski and Gal (2017) train a network for masking salient regions.

Input Reconstruction As our method synthesizes inputs to the agent, the most closely related work includes input reconstruction techniques. Long et al. (2014) reconstruct an image from an average of image patches based on nearest neighbors in feature space. Mahendran and Vedaldi (2015) propose to reconstruct images by inverting representations learned by CNNs, while Dosovitskiy and Brox (2015) train a CNN to reconstruct the input from its encoding.

When maximizing the activation of a specific class or neuron, regularization is crucial because the optimization procedure—starting from a random noise image and maximizing an output—is vastly under-constrained and often tends to generate fooling examples that fall outside the manifold of realistic images (Nguyen et al., 2015). In Mahendran and Vedaldi (2016) total variation (TV) is used for regularization, while Baust et al. (2018) propose an update scheme based on Sobolev gradients. In Nguyen et al. (2015) Gaussian filters are used to blur the pre-image or the update computed in every iteration. Since there are usually multiple input families that excite a neuron, Nguyen et al. (2016c) propose an optimization scheme for the distillation of these clusters. Ulyanov et al. (2017) show that even CNNs with random weights can be used for regularization. More variations of regularization can be found in Olah et al. (2017; 2018). Instead of regularization, Nguyen et al. (2016a;b) use a denoising autoencoder and optimize in latent space to reconstruct pre-images for image classification.

2.2 Explanations for Reinforcement Learning

In deep reinforcement learning however, feature visualization is to date relatively unexplored. Zahavy et al. (2016) apply t-SNE (Maaten and Hinton, 2008) on the last layer of a deep Q-network (DQN) to

cluster states of behavior of the agent. Mnih et al. (2016) also use t-SNE embeddings for visualization, while Greydanus et al. (2017) examine how the current state affects the policy in a vision-based approach using *saliency methods*. Wang et al. (2016) use saliency methods from Simonyan et al. (2013) to visualize the value and advantage function of their dueling Q-network. Interestingly, we could not find prior work using *activation maximization* methods for visualization. In our experiments we show that the typical methods fail in the case of RL networks and generate images far outside the manifold of valid game states, even with all typical forms of regularization. In the next section, we will show how to overcome these difficulties.

3 METHODS

We will first introduce the notation and definitions that will be used through out the remainder of the paper. We formulate the reinforcement learning problem as a discounted, infinite horizon Markov decision process $(\mathcal{S}, \mathcal{A}, \gamma, P, r)$, where at every time step t the agent finds itself in a state $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$ following its policy $\pi_\theta(a|s_t)$. Then the environment transitions from state s_t to state s_{t+1} given the model $P(s_{t+1}|s_t, a_t)$. Our goal is to visualize RL agents given a user-defined objective function, without adding constraints on the optimization process of the agent itself, i.e. assuming that we are given a previously trained agent with fixed parameters θ.

We approach visualization via a generative model over the state space $\mathcal{S}$ and synthesize states that lead to an interesting, user-specified behavior of the agent. This could be, for instance, states in which the agent expresses high uncertainty regarding which action to take or states in which it sees no good way out. This approach is fundamentally different than saliency-based methods as they always need an input for the test-set on which the saliency maps can be computed. The generative model constrains the optimization of states to induce specific agent behavior.

3.1 STATE MODEL

Often in feature visualization for CNNs, an image is optimized starting from random noise. However, we found this formulation too unconstrained, often ending up in local minima or fooling examples (Figure 4a). To constrain the optimization problem we learn a generative model on a set $\mathcal{S}$ of states generated by the given agent that is acting in the environment. The model is inspired by variational autoencoders (VAEs) (Kingma and Welling, 2013) and consists of an encoder $f(s) = (\mu, \sigma) \in \mathbb{R}^{2\times n}$ that maps inputs to a Gaussian distribution in latent space and a decoder $g(\mu, \sigma, z) = \hat{s}$ that reconstructs the input. The training of our generator has three objectives. First, we want the generated samples to be close to the manifold of valid states s. To avoid fooling examples, the samples should also induce correct behavior in the agent and lastly, sampling states needs to be efficient. We encode these goals in three corresponding loss terms.

$$\mathcal{L}(s) = \mathcal{L}_p(s) \; + \eta \mathcal{L}_a(s) \; + \lambda \mathcal{KL}(\, f(s), \mathcal{N}(0, I_n)\,). \tag{1}$$

The role of $\mathcal{L}_p(s)$ is to ensure that the reconstruction $g(f(s), z)$ is close to the input s such that $\| \, g(f(s), z) - s \, \|_2^2$ is minimized. We observe that in the typical reinforcement learning benchmarks, such as Atari games, small details—e.g. the ball in Pong or Breakout—are often critical for the decision making of the agent. However, a typical VAE model tends to yield blurry samples that are not able to capture such details. To address this issue, we model the reconstruction error $\mathcal{L}_p(s)$ with an *attentive loss* term, which leverages the saliency of the agent to put focus on critical regions of the reconstruction. The saliency maps are computed by guided backpropagation of the policy's gradient with respect to the state.

$$\mathcal{L}_p(s) = \| \, g(f(s), z) - s \, \|_2^2 \; \odot \frac{\| \, \nabla \pi(s) \, \|_1}{\sum_{i=1}^{d} \| \, \nabla \pi(s)_i \, \|_1}. \tag{2}$$

Since we are interested in the actions of the agent on synthesized states, the second objective $\mathcal{L}_a(s)$ is used to model the *perception of the agent*:

$$\mathcal{L}_a(s) = \| \, A(s) - A(\, g(f(s), z)\,) \, \|_2^2, \tag{3}$$

where A is a generic formulation of the output of the agent. For a DQN for example, $\pi(s) = \max_a A(s)_a$, i.e. the final action is the one with the maximal Q-value. This term encourages the

reconstructions to be interpreted by the agent the same way as the original inputs s. The last term $\mathcal{KL}(\, f(s), \mathcal{N}(0, I_n)\,)$ ensures that the distribution predicted by the encoder f stays close to a Gaussian distribution. This allows us to initialize the optimization with a reasonable random vector later and forms the basis of a regularizer. Thus, after training, the model approximates the distribution of states $p(s)$ by sampling z from $\mathcal{N}(0, I_n)$. We will now use the generator inside an optimization scheme to generate state samples that satisfy a user defined target objective.

3.2 Sampling States of Interest

Training a generator with the objective function of Equation 1 allows us to sample states that are not only visually close to the real ones, but which the agent can also interpret and act upon as if they were states from a real environment.

We can further exploit this property and formulate an energy optimization scheme to generate samples that satisfy a specified objective. The energy operates on the latent space of the generator and is defined as the sum of a target function T on agent's policy and a regularizer R

$$E(x) = T(\, \pi(\, g(x, z)\,)\,) + \alpha R(x). \tag{4}$$

The target function can be defined freely by the user and depends on the agent that is being visualized. For a DQN, one could for example define T as the Q-value of a certain action, e.g. pressing the brakes of a car. In section 3.3, we show several examples of targets that are interesting to analyze. The regularizer R can again be chosen as the KL divergence between x and the normal distribution:

$$R(x) = \mathcal{KL}(\, x, \mathcal{N}(0, I_n)\,), \tag{5}$$

forcing the samples that are drawn from the distribution x to be close to the Gaussian distribution that the generator was trained with. We can optimize Equation 4 with gradient descent on $x = (\sigma, \mu)$.

3.3 Target Functions

Depending on the agent, one can define several interesting target functions T. For a DQN the previously discussed action maximization is interesting to find situations in which the agent assigns a high value to a certain action e.g. $T_{left}(s) = -A_{left}(s)$. Other states of interest are those to which the agent assigns a low (or high) value for all possible actions $A(s) = q = (q_1, \ldots, q_m)$. Consequently, one can optimize towards a low Q-value for the highest valued action with the following objective:

$$T^{-}(q) = \frac{\sum_{i=1}^{m} q_i e^{\beta q_i}}{\sum_{k=1}^{m} e^{\beta q_k}}, \tag{6}$$

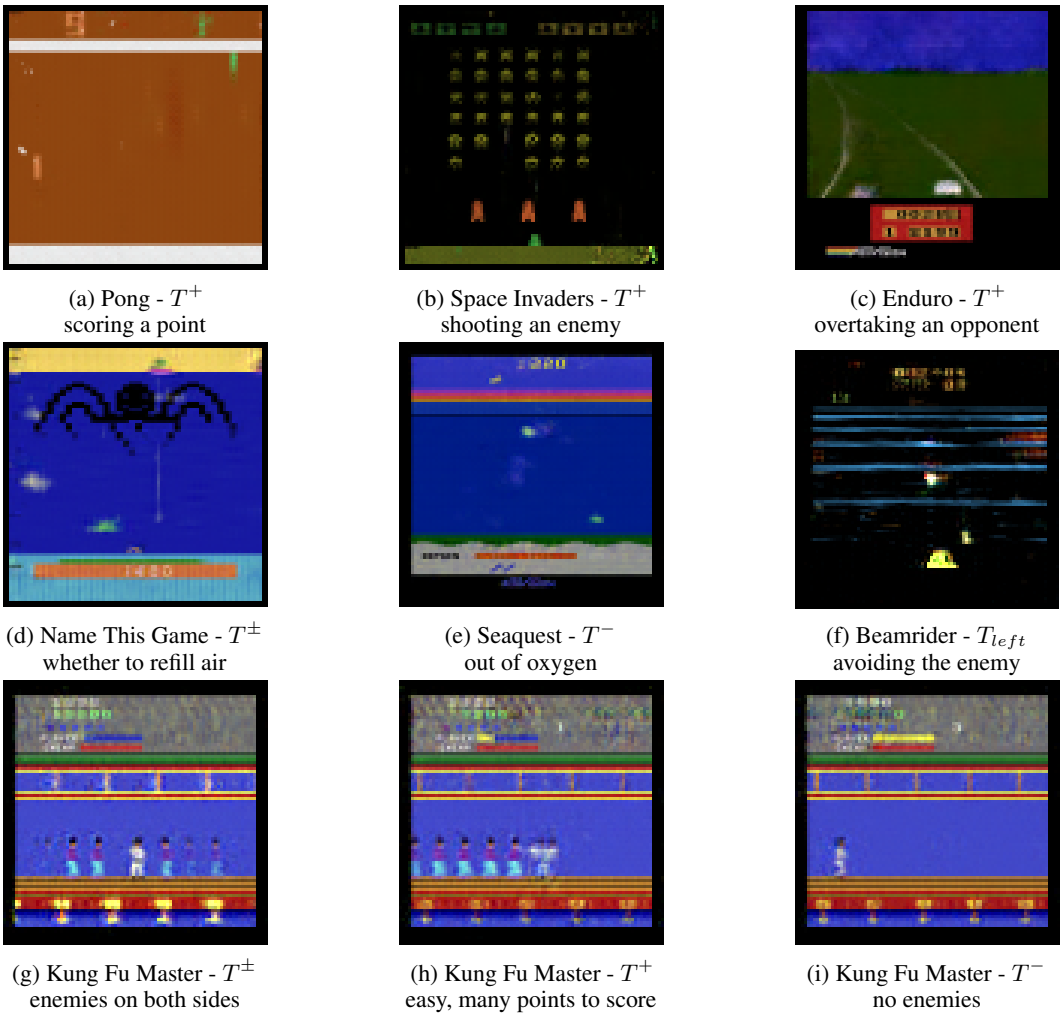
where $\beta > 0$ controls the sharpness of the soft maximum formulation. Analogously, one can maximize the lowest Q-value with $T^{+}(q) = -T^{-}(-q)$. We can also optimize for interesting situations in which one action is of very high value and another is of very low value by defining

$$T^{\pm}(q) = T^{-}(q) - T^{+}(q). \tag{7}$$

4 Experiments

In this section we thoroughly evaluate and analyze our method on Atari games (Bellemare et al., 2013) using the OpenAI Gym (Brockman et al., 2016) and a driving simulator. We present qualitative results for three different reinforcement learning algorithms, show examples on how the method helps finding flaws in an agent, analyze the loss contributions and compare to previous techniques.

Implementation details In all our experiments we use the same factors to balance the loss terms in Equation 6: $\lambda = 10^{-4}$ for the KL divergence and $\eta = 10^{-3}$ for the agent perception loss. The generator is trained on $10,000$ frames (using the agent and an ϵ-greedy policy with $\epsilon = 0.1$). Optimization is done with Adam (Kingma and Ba, 2015) with a learning rate of 10^{-3} and a batch size of 16 for 2000 epochs. Training takes approximately four hours on a Titan Xp. Our generator uses a latent space of 100 dimensions, and consists of four encoder stages comprised of a 3×3 convolution with stride 2, batch-normalization (Ioffe and Szegedy, 2015) and ReLU layer. The starting number of



(a) Pong - T^+ scoring a point

(b) Space Invaders - T^+ shooting an enemy

(c) Enduro - T^+ overtaking an opponent

(d) Name This Game - $T^\pm$ whether to refill air

(e) Seaquest - T^- out of oxygen

(f) Beamrider - T_{left} avoiding the enemy

(g) Kung Fu Master - $T^\pm$ enemies on both sides

(h) Kung Fu Master - T^+ easy, many points to score

(i) Kung Fu Master - T^- no enemies

Figure 1: **Qualitative Results:** Visualization of different target functions (Sec. 3.3). T^+ generates high reward and T^- low reward states; $T^\pm$ generates states in which one action is highly beneficial and another is bad.

filters is 32 and is doubled at every stage. A fully connected layer is used for mean and log-variance prediction. Decoding is inversely symmetric to encoding, using deconvolutions and halving the number of channels at each of the four steps.

For the experiments on the Atari games we train a double DQN (Wang et al., 2016) for two million steps with a reward discount factor of 0.95. The input size is 84×84 pixels. Therefore, our generator performs up-sampling by factors of 2, up to a 128×128 output, which is then center cropped to 84×84 pixels. The agents are trained on grayscale images, for better visual quality however, our generator is trained with color frames and convert to grayscale using a differentiable, weighted sum of the color channels. In the interest of reproducibility we will make the visualization code available.

4.1 Visualizations On Atari Games

In Figure 1, we show qualitative results from various Atari games using different target functions T, as described in Section 3.3. From these images we can validate that the general visualizations that are obtained from the method are of good quality and can be interpreted by a human. T^+ generates generally high value states independent of a specific action (first row of Figure 1), while T^- generates low reward situations, such as close before losing the game in Seaquest (Figure 1.e) or when there are no points to score (Figure 1.i). Critical situations can be found by maximizing the difference

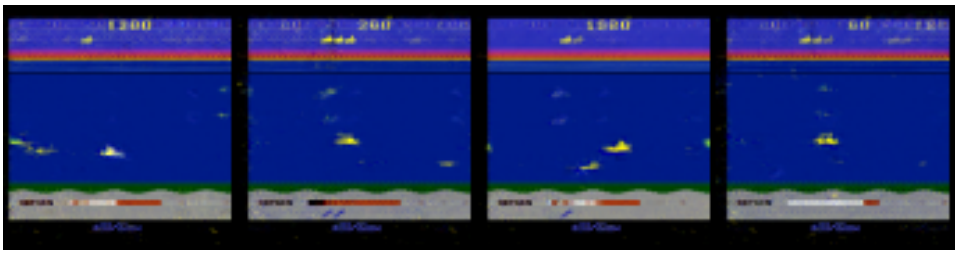


Figure 2: **Seaquest with ACKTR.** Visualization results for a network trained with ACKTR on Seaquest. The objective is $T^{\pm}$ indicating situations that can be rewarding but also have a low scoring outcome. The generated states show low oxygen or close proximity to enemies.

between lowest and highest estimated Q-value with $T^{\pm}$. In those cases, there is clearly a right and a wrong action to take. In Name This Game (Figure 1.d) this occurs when close to the air refill pipe, which prevents suffocating under water; in Kung Fu Master when there are enemies coming from both sides (Figure 1.g), the order of attack is critical, especially since the health of the agent is low (yellow/blue bar on top). An example of maximizing the value of a single action (similar to maximizing the confidence of a class when visualizing image classification CNNs) can be seen in (Figure 1.f) where the agent sees moving left and avoiding the enemy as the best choice of action.

4.2 ACKTR

To show that this visualization technique generalizes over different RL algorithms, we also visualize ACKTR (Wu et al., 2017). We use the code and pretrained models from a public repository (Kostrikov, 2018) and train our generative model with the same hyperparameters as above and without any modifications on the agent. We present the $T^{\pm}$ objective for the ACKTR agent in Figure 2 to visualize states with both high and low rewards, for example low oxygen (surviving vs. suffocating) or close proximity to enemies (earning points vs. dying). Compared to the DQN visualizations the ACKTR visualizations, are almost identical in terms of image quality and interpretability. This supports the notion that our proposed approach is independent of the specific RL algorithm.

4.3 Interpretation of Visualizations

Analyzing the visualizations on Seaquest, we make an interesting observation. When maximizing the Q-value for the actions, in many samples we see a low or very low oxygen meter. In these cases the submarine would need to ascend to the surface to avoid suffocation. Although the up action is the only sensible choice in this case, we also obtain visualized low oxygen states for all other actions. This implies that the agent has not understood the importance of resurfacing when the oxygen is low. We then run several roll outs of the agent and see that the major cause of death is indeed suffocation and not collision with enemies. This shows the impact of visualization, as we are able to understand a flaw of the agent. Although it would be possible to identify this flawed behavior directly by analyzing the $10,000$ frames of training data for our generator, it is significantly easier to review a handful of samples from our method. Further, as the generator is a generative model, we can synthesize states that are not part of its training set.

4.4 Ablation Studies (Loss Terms)

In this section we analyze the three loss terms of our generative model. The human perceptual loss is weighted by the (guided) gradient magnitude of the agent in Equation 2. In Figure 3 we visualize this mask for a DQN agent for random frames from the dataset. The masks are blurred with an averaging filter of kernel size 5. We observe that guided backpropagation results in precise saliency maps focusing on player and enemies that then focus the reconstructions on what is important for the agent.

To study the influence of the loss terms we perform an experiment in which we evaluate the agent not on the real frames but on their reconstructions. If the reconstructed frames are perfect, the agent with generator goggles achieves the same score as the original agent. We can use this metric to understand the quantitative influence of the loss terms. In Pong, the ball is the most important visual aspect of

Table 1: **Loss Study.** We compare the performance of the original agent with the agent operating on reconstructed frames instead. The original performance represents an upper bound for the score of the same agent which is operating on reconstructions instead. Shown are average scores over 20 runs.

	Agent	VAE baseline	Ours ($\mathcal{L}_p$ only)	Ours (full)
Pong	14	-8	4	14
Atlantis	108	95	98	109
Q*bert	64	26	28	31

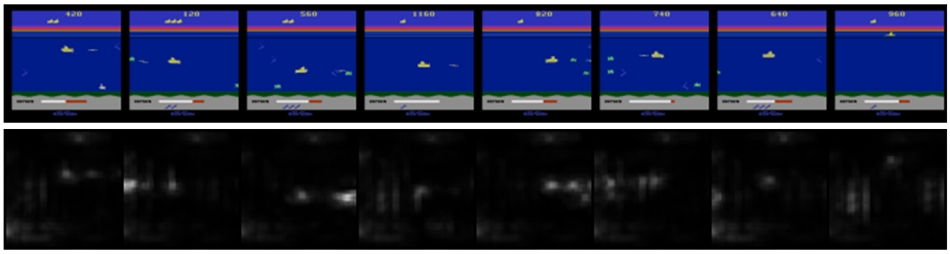


Figure 3: **Weight Visualization.** We visualize the weighting (second row) of the reconstruction loss from Equation 2 for eight randomly drawn samples (first row) of the dataset. Most weight lies on the player's submarine and close enemies, supporting their importance for the decision making.

the game for decision making. In Table 1 we see that the VAE baseline scores much lower than our model. Since the ball is very small, it is mostly ignored by the reconstruction loss of a VAE. Our formulation is built to regain the original performance of the agent. Overall, we see that our method always improves over the baseline but does not always match the original performance.

4.5 Comparison with Activation Maximization

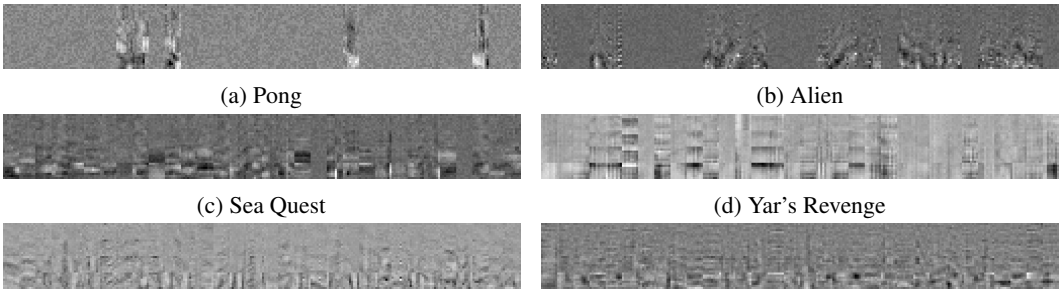
For image classification tasks, activation maximization works well when optimizing the pre-image directly (Mahendran and Vedaldi, 2015; Baust et al., 2018). However we find that for reinforcement learning, the features learned by the network are not complex enough to reconstruct meaningful pre-images, even with sophisticated regularization techniques. The pre-image converges to a *fooling example* maximizing the class but being far away from the manifold of states of the environment.

In Figure 4.a we compare our results with the reconstructions generated using the method of Baust et al. (2018) for a DQN agent. We obtain similarly bad pre-images with TV-regularization (Mahendran and Vedaldi, 2016), Gaussian blurring (Nguyen et al., 2015) and other regularization tricks such as random jitter, rotations, scaling and cropping (Olah et al., 2017). One explanation for the low performance of standard methods for activation maximization can be found when visualizing the first layer filters of Atari agents. We show Conv1 of a DQN in Figure 5. We stack the representations for the temporal component vertically and the 32 filters horizontally. Looking at the filters of Pong, we make two observations. The agent only needs five distinct filters to play the game and due to the strong temporal changes in patterns, it is mostly focused on moving parts. This aligns with the reactive game play of pong and its visual simplicity. Instead, a complex game such as Seaquest uses all available filters.

These observations bring up interesting points. Indeed the CNN architecture of Mnih et al. (2015) contains enough filters for the most visually complex games that are not needed for the simpler environments such as Pong. Also, the temporal component seems to only be important for some of the environments. This, and the strong visible differences between the weights of different environments leaves the open question whether a common feature extractor exists that could work for all games. However, poor Conv1 weights are an indicator that it is easier to distract the model by activating "unused" filters with imperceptible noise which can be the cause for the poor visualizations with classical activation maximization techniques.

(a) Activation Maximization (b) Ours

Figure 4: **Comparison with activation maximization.** The visual features learned by the agents are not complex enough to reconstruct typical frames from the game via activation maximization. This problem is mitigated in our method by learning a low-dimensional embedding of games states first.



(a) Pong (b) Alien

(c) Sea Quest (d) Yar's Revenge

(e) Krull (f) Name This Game

Figure 5: **Conv1 Weights DQN.** We display the weights of the first layer of DQNs trained on several Atari games. Columns represent the 32 different 8×8 filters, while the four rows correspond the to four frames that are stacked as an input for the network. The vertically stacked weights represent the temporal component with the current frame at the bottom and frame $t - 3$ at the top.

4.6 Experiments with a Driving Simulator

We have created a 3D driving simulation environment and trained an A2C agent maximizing speed while avoiding pedestrians that are crossing the road. The agent is trained with four temporal semantic segmentation frames (128×128 pixels) as input (Figure 6). With this safety-critical application we can assess multiple points. First, driving is a continuous task in a much more complex environment than Atari games. Second, we use the simulator to build two custom environments and validate that we can identify problematic behavior in the agent. Specifically, we train the agent in a "*reasonable pedestrians*" environment, where pedestrians cross the road carefully, when no car is coming or at traffic lights. With these choices, we model data collected in the real world, where it is unlikely that people unexpectedly run in front of the car. We visualize states in which the agent expects a low future return (T^- objective) in Figure 6. It shows that the agent is aware of other cars, traffic lights and intersections. However, there are no generated states in which the car is about to collide with a person, meaning that the agent does not recognize the criticality of pedestrians. To verify our suspicion, we test this agent in a "*distracted pedestrians*" environment where people cross the road looking at their phones without paying attention to approaching cars. We find that the agent does indeed run over humans. With this experiment, we show that our visualization technique can identify biases in the training data just by critically analyzing the sampled frames.

While one could simply examine the experience replay buffer to find scenarios of interest, our approach allows unseen scenarios to be synthesized. To quantitatively evaluate the assertion that our generator is capable of generating novel states, we sample states and compare them to their closest frame in the training set under an MSE metric. We count a pixel as different if the relative difference in a channel exceeds 25% and report the histogram in Table 2. The results show that there are very few samples that are very close to the training data. On average a generated state is different in 25% of the pixels, which is high, considering the overall common layout of the road, buildings and sky.

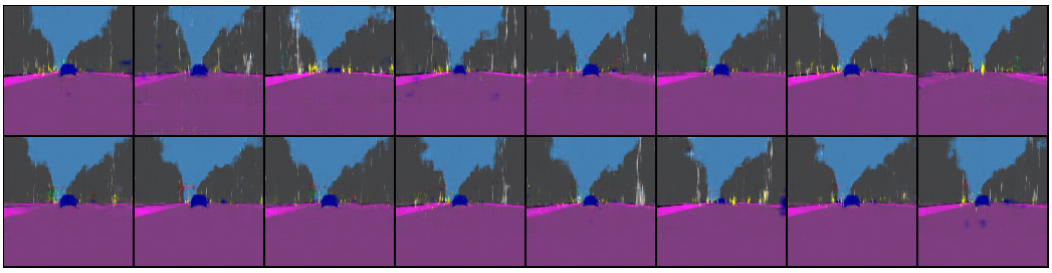


Figure 6: **Driving simulator.** We show 16 samples for the T^- objective of an agent trained in the *reasonable pedestrians* environment. From these samples one can infer that the agent is aware of traffic lights (red) and other cars (blue) but has very likely not understood the severity of hitting pedestrians (yellow). Deploying this agent in the *distracted pedestrians* environment shows that the agent indeed collides with people that cross the road in front of the agent.

Table 2: **Synthesizing unseen states.** We compare generated samples to their closest neighbor in the training set and compute the percentage of pixels whose values differ by at least 25%, e.g. 73% of the synthesized samples differ in more than 20% pixels in comparison to their closest training sample.

#pixels different	$> 10\%$	$> 20\%$	$> 30\%$	$> 40\%$	$> 50\%$	$> 60\%$	$> 70\%$
samples	99%	73%	16%	4%	1%	1%	0%

5 Discussion and Conclusions

We have presented a method to synthesize inputs to deep reinforcement learning agents based on generative modeling of the environment and user-defined objective functions. Training the generator to produce states that the agent perceives as those from the real environment enables optimizing its latent space to sample states of interest. We believe that understanding and visualizing agent behavior in safety critical situations is a crucial step towards creating safer and more robust agents using reinforcement learning. We have found that the methods explored here can indeed help accelerate the detection of problematic situations for a given learned agent. As such we intend to build upon this work.

References

Adebayo, J., Gilmer, J., Goodfellow, I., and Kim, B. (2018). Local explanation methods for deep neural networks lack sensitivity to parameter values. *International Conference on Learning Representations, Workshop Track*.

Baust, M., Ludwig, F., Rupprecht, C., Kohl, M., and Braunewell, S. (2018). Understanding regularization to visualize convolutional neural networks. *arXiv preprint arXiv:1805.00071*.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Chattopadhay, A., Sarkar, A., Howlader, P., and Balasubramanian, V. N. (2018). Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 839–847. IEEE.

Dabkowski, P. and Gal, Y. (2017). Real time image saliency for black box classifiers. In *Advances in Neural Information Processing Systems*, pages 6970–6979.

Dosovitskiy, A. and Brox, T. (2015). Inverting convolutional networks with convolutional networks. *CoRR abs/1506.02753*.

Erhan, D., Bengio, Y., Courville, A., and Vincent, P. (2009). Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1.

Fong, R. C. and Vedaldi, A. (2017). Interpretable explanations of black boxes by meaningful perturbation. *arXiv preprint arXiv:1704.03296*.

Greydanus, S., Koul, A., Dodge, J., and Fern, A. (2017). Visualizing and understanding atari agents. *arXiv preprint arXiv:1711.00138*.

Grün, F., Rupprecht, C., Navab, N., and Tombari, F. (2016). A taxonomy and library for visualizing learned features in convolutional neural networks. *arXiv preprint arXiv:1606.07757*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

Karpathy, A., Johnson, J., and Fei-Fei, L. (2016). Visualizing and understanding recurrent networks. *International Conference on Learning Representations*.

Kindermans, P.-J., Hooker, S., Adebayo, J., Alber, M., Schütt, K. T., Dähne, S., Erhan, D., and Kim, B. (2017a). The (un) reliability of saliency methods. *arXiv preprint arXiv:1711.00867*.

Kindermans, P.-J., Schütt, K. T., Alber, M., Müller, K.-R., and Dähne, S. (2017b). Patternnet and patternlrp–improving the interpretability of neural networks. *arXiv preprint arXiv:1705.05598*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kostrikov, I. (2018). Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr`.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

Long, J. L., Zhang, N., and Darrell, T. (2014). Do convnets learn correspondence? In *Advances in Neural Information Processing Systems*, pages 1601–1609.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Mahendran, A. and Vedaldi, A. (2015). Understanding deep image representations by inverting them. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 5188–5196. IEEE.

Mahendran, A. and Vedaldi, A. (2016). Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120(3):233–255.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T., and Clune, J. (2016a). Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems*, pages 3387–3395.

Nguyen, A., Yosinski, J., Bengio, Y., Dosovitskiy, A., and Clune, J. (2016b). Plug & play generative networks: Conditional iterative generation of images in latent space. *arXiv preprint arXiv:1612.00005*.

Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436.

Nguyen, A., Yosinski, J., and Clune, J. (2016c). Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *arXiv preprint arXiv:1602.03616*.

Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*. https://distill.pub/2017/feature-visualization.

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. (2018). The building blocks of interpretability. *Distill*. https://distill.pub/2018/building-blocks.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2016). Grad-cam: Visual explanations from deep networks via gradient-based localization. *See https://arxiv. org/abs/1610.02391 v3*, 7(8).

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2017). Deep image prior. *arXiv preprint arXiv:1711.10925*.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, pages 1995–2003. JMLR. org.

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5285–5294.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057.

Zahavy, T., Ben-Zrihem, N., and Mannor, S. (2016). Graying the black box: Understanding dqns. In *International Conference on Machine Learning*, pages 1899–1908.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

Zhang, J., Lin, Z., Brandt, J., Shen, X., and Sclaroff, S. (2016). Top-down neural attention by excitation backprop. In *European Conference on Computer Vision*, pages 543–559. Springer.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2016). Learning deep features for discriminative localization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zintgraf, L. M., Cohen, T. S., Adel, T., and Welling, M. (2017). Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*.

APPENDIX

To show an unbiased and wide variety of results, in the following, we will show four random samples generated by our method for a DQN agent trained on many of the Atari benchmark environments. We show visualizations optimized for a meaningful objective for each game (e.g. not optimizing for unused buttons). All examples were generated with the same hyperparameter settings.

Please note that for some games better settings can be found. Some generators on visually more complex games would benefit from longer training to generate sharper images. Our method is able to generate reasonable images even when the DQN was unable to learn a meaningful policy such as for *Montezuma's revenge*. We show two additional objectives maximizing/minimizing the expected reward of the state under a random action: $S^+(q) = \sum_{i=1}^{m} q_i$ and $S^-(q) = -S^+(q)$. Results in alphabetical order.

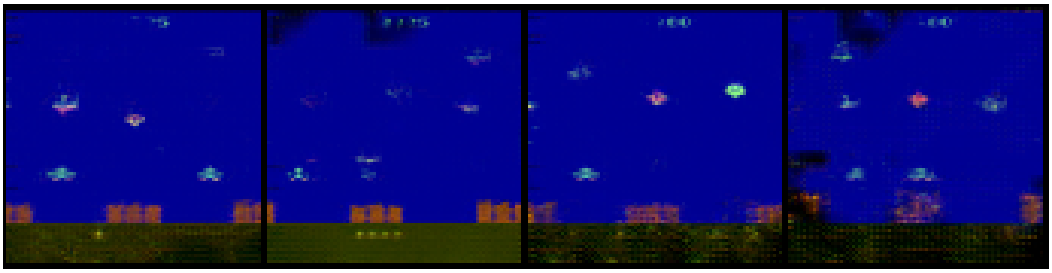


Figure 7: **Air Raid.** Target function: S^+.

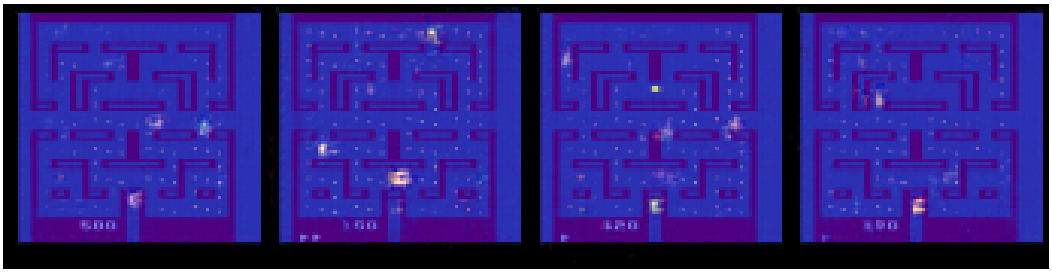


Figure 8: **Alien.** Target function: right.

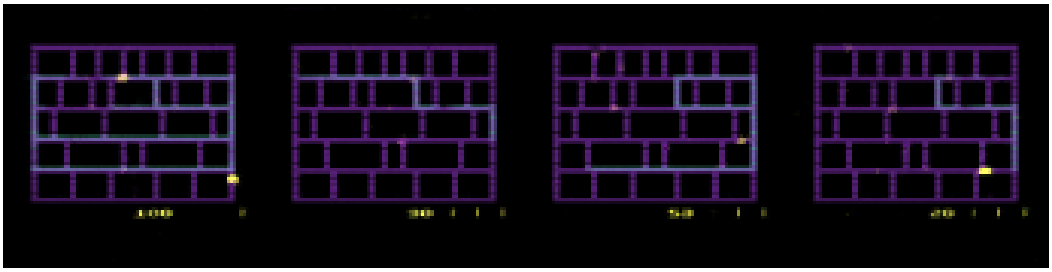


Figure 9: **Amidar.** Target function: up.

Figure 10: **Assault.** Target function: S^-.

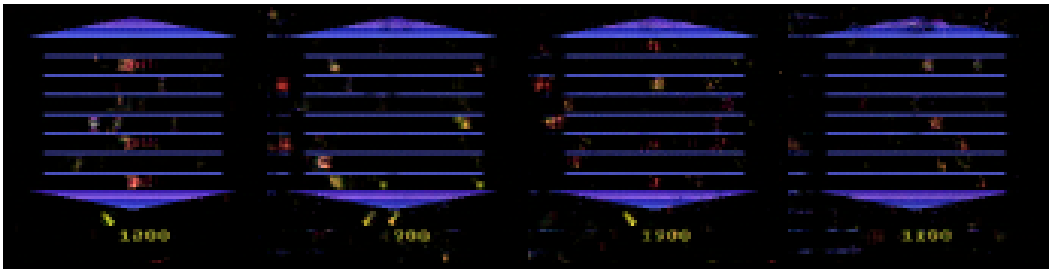


Figure 11: **Asterix.** Target function: T^-.

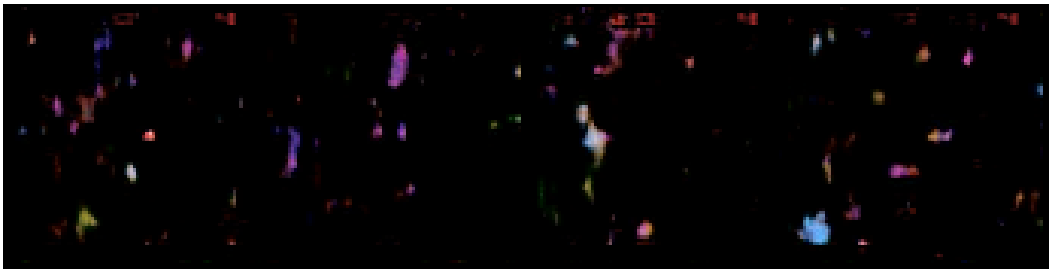


Figure 12: **Asteroids.** Target function: up-fire.

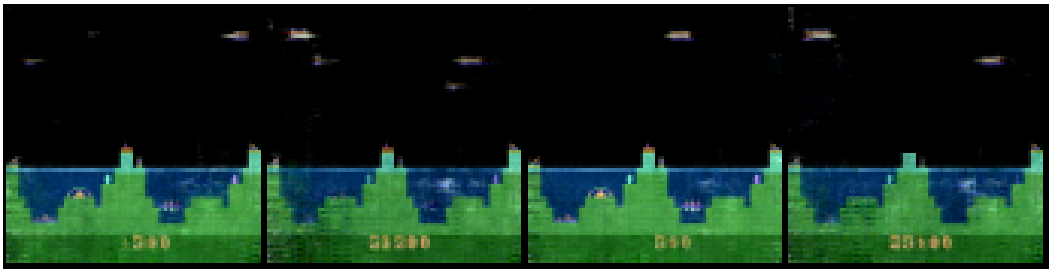


Figure 13: **Atlantis.** Target function: T^+.

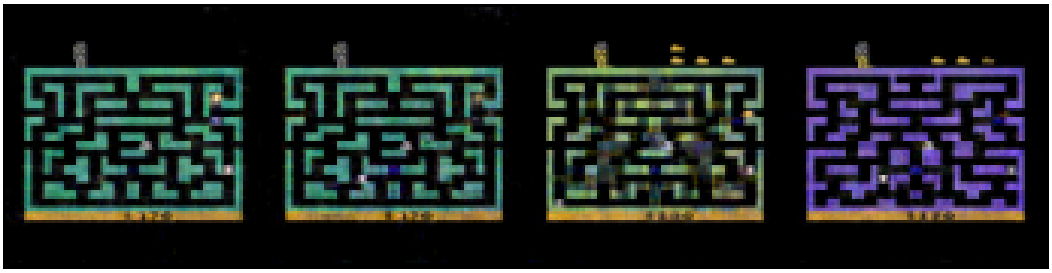


Figure 14: **Bank Heist.** Target function: T^+.

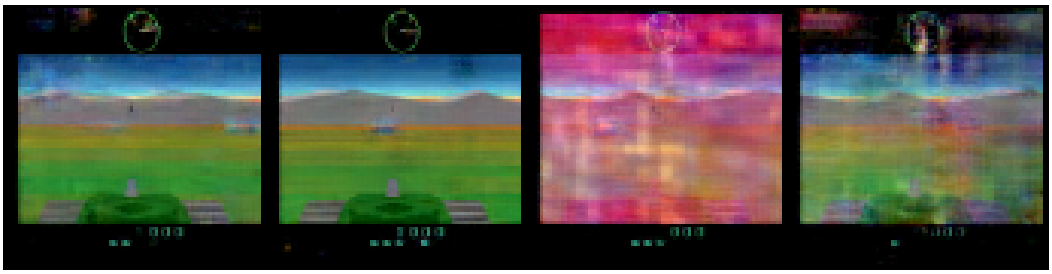


Figure 15: **Battlezone.** Target function: T^-.

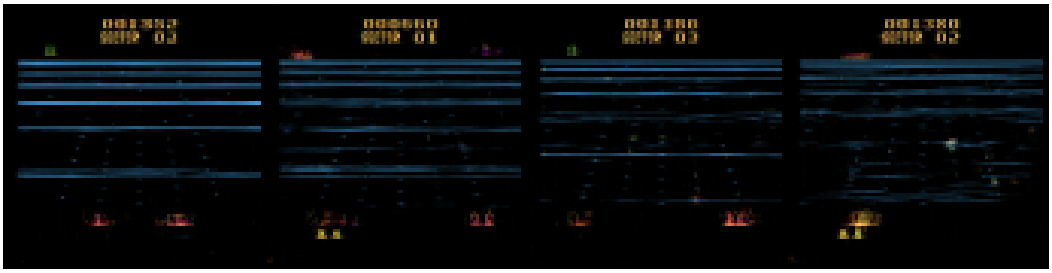


Figure 16: **Beamrider.** Target function: T^+.

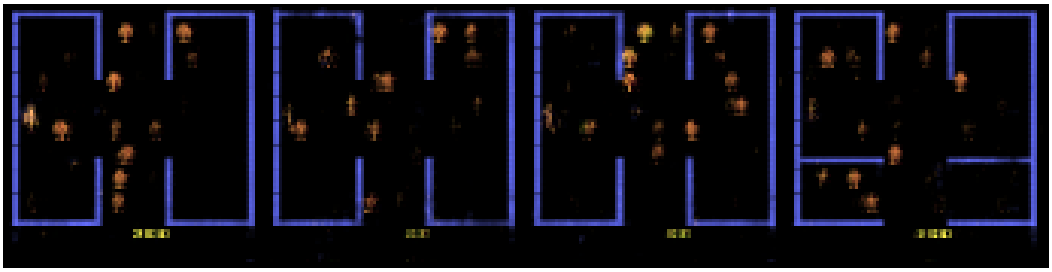


Figure 17: **Berzerk.** Target function: S^+.

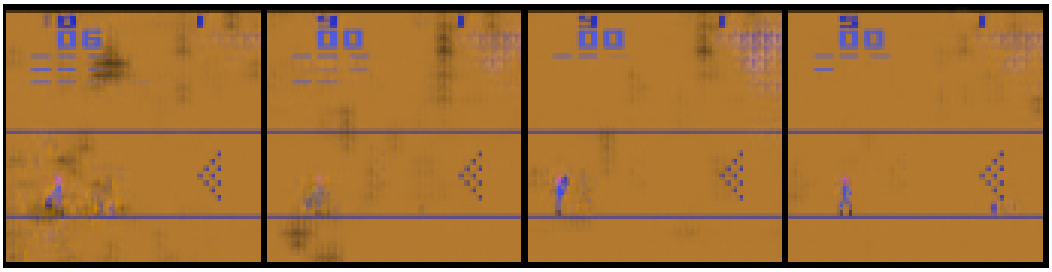


Figure 18: **Bowling.** Target function: S^+.

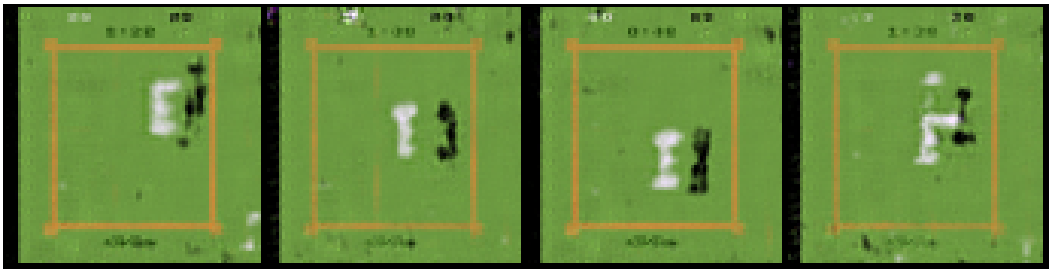


Figure 19: **Boxing.** Target function: S^+.

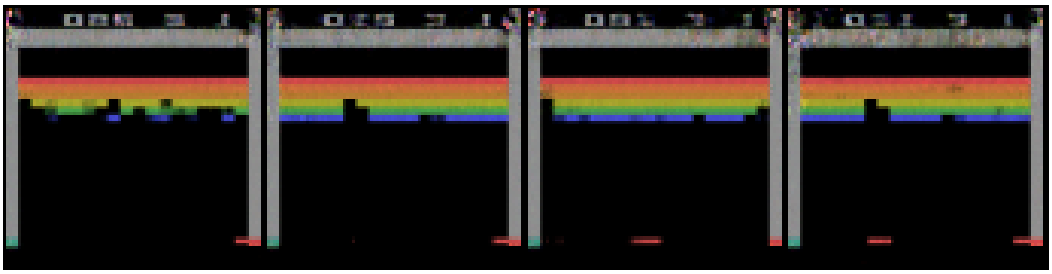


Figure 20: **Breakout.** Target function: T^{-}.

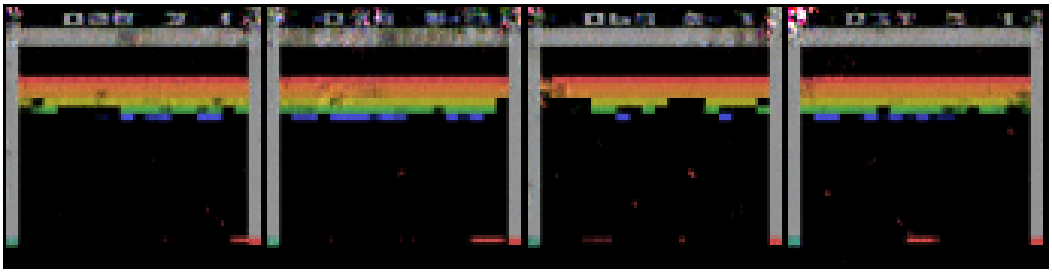


Figure 21: **Breakout.** Target function: Left.

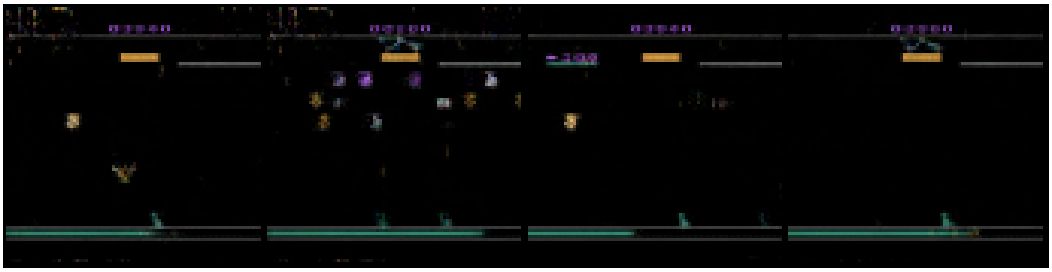


Figure 22: **Carnival.** Target function: right.

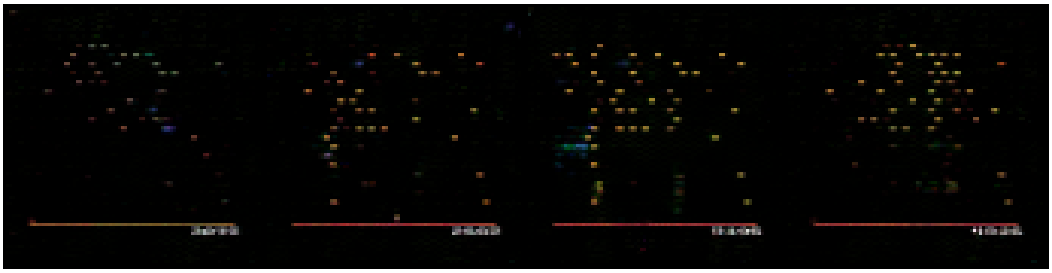


Figure 23: **Centipede.** Target function: $T^{\pm}$.

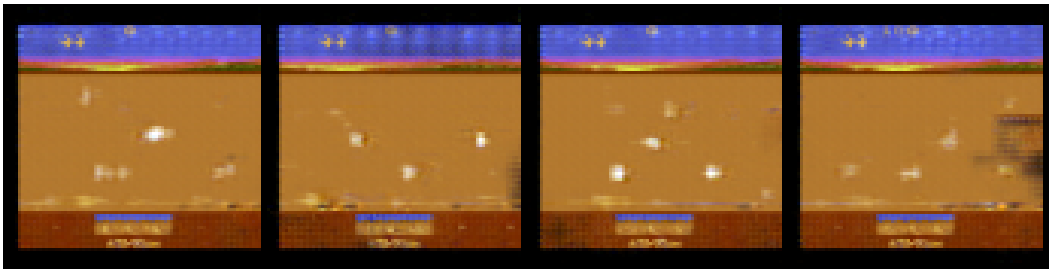


Figure 24: **Chopper Command.** Target function: S^{+}.

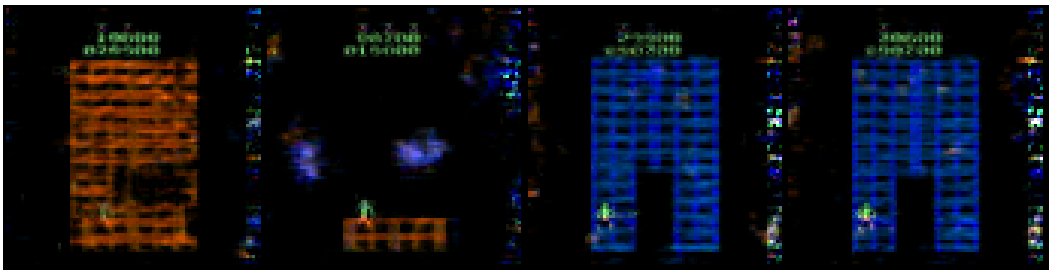


Figure 25: **Crazy Climber.** Target function: T^-.

Figure 26: **Demon Attack.** Target function: T^+.

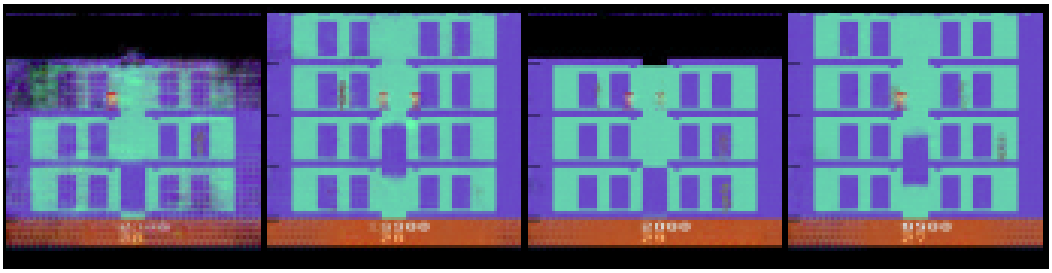


Figure 27: **Elevator Action.** Target function: no-op.

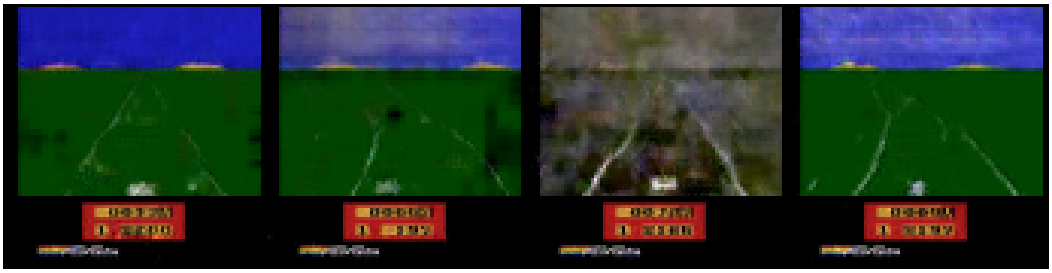


Figure 28: **Enduro.** Target function: S^+.

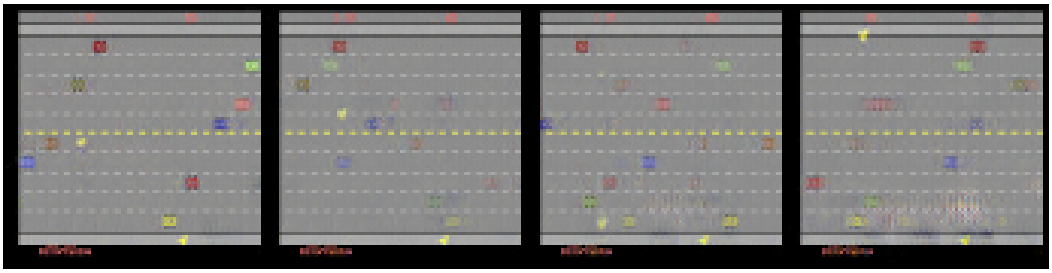


Figure 29: **Freeway.** Target function: T^+.

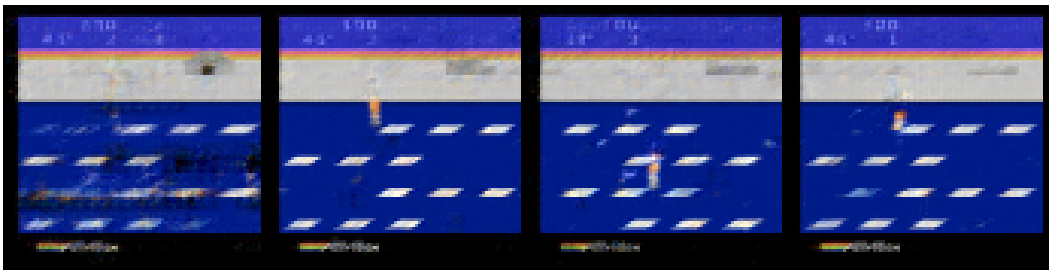


Figure 30: **Frostbite.** Target function: no-op.

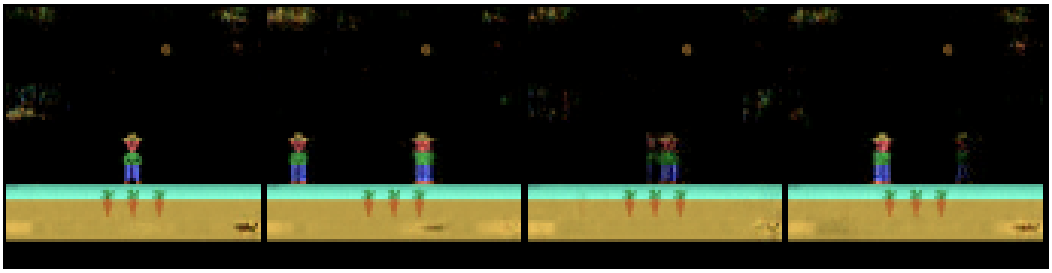


Figure 31: **Gopher.** Target function: S^{-}.

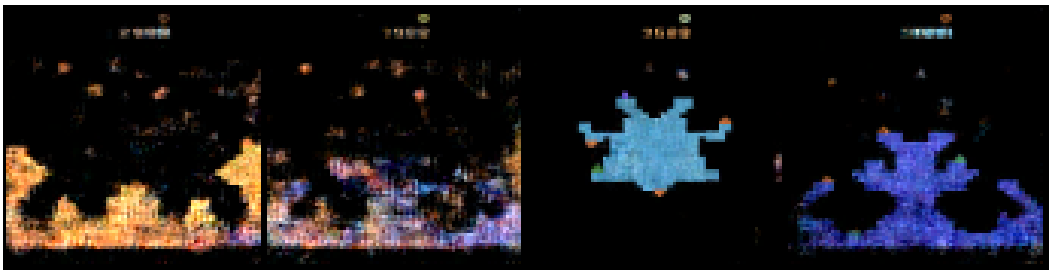


Figure 32: **Gravitar.** Target function: $T^{\pm}$.

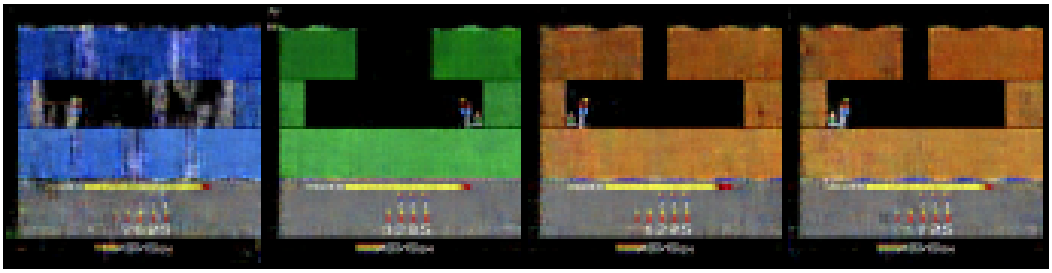


Figure 33: **Hero.** Target function: S^{+}.

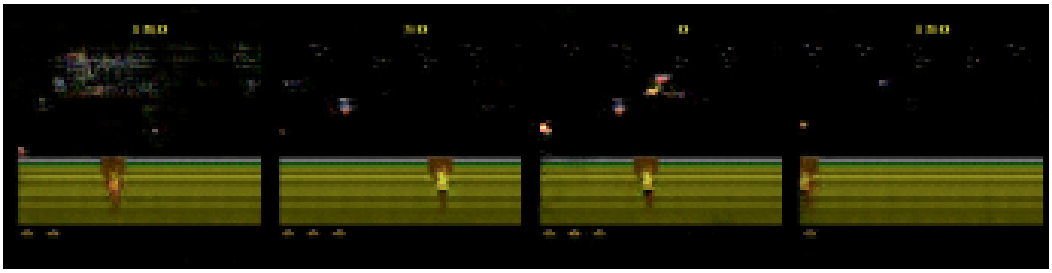


Figure 34: **JamesBond.** Target function: S^{+}.

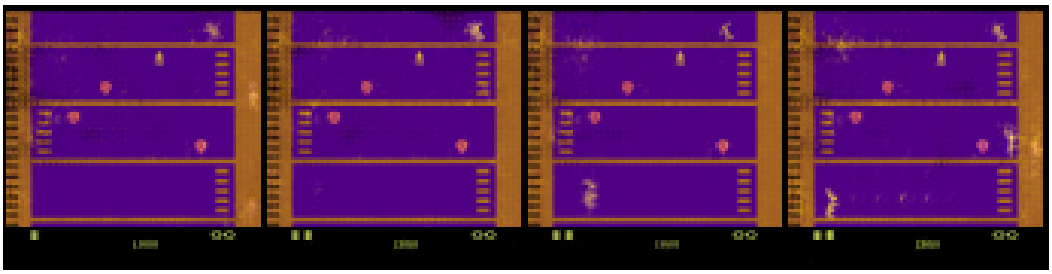


Figure 35: **Kangaroo.** Target function: S^-.

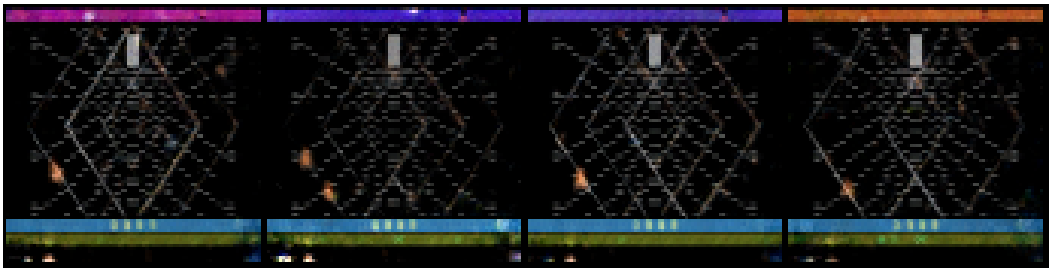


Figure 36: **Krull.** Target function: fire.

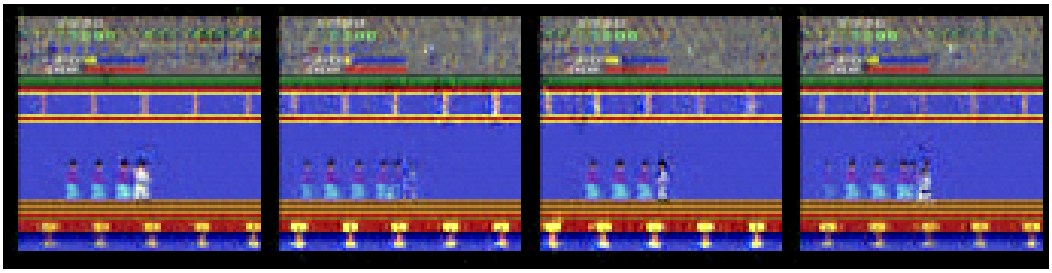


Figure 37: **Kung Fu Master.** Target function: up.

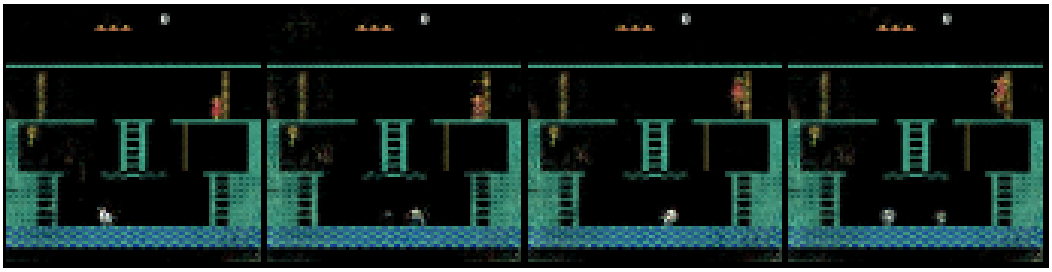


Figure 38: **Montezuma's Revenge.** Target function: T^-.

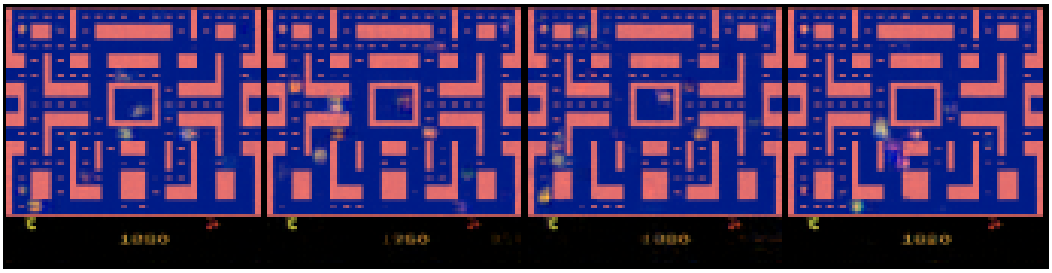


Figure 39: **Ms. Pacman.** Target function: no-op.

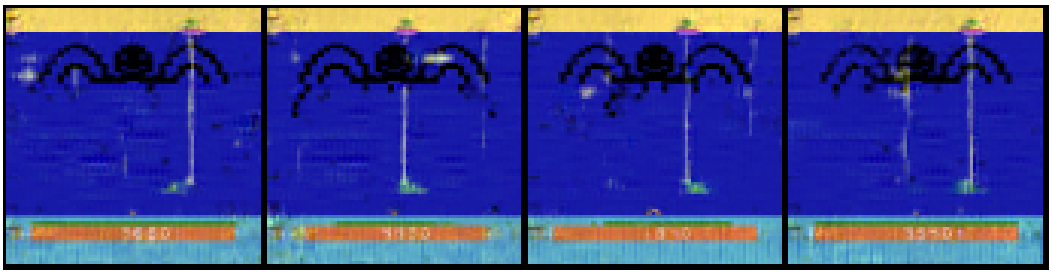


Figure 40: **Name This Game.** Target function: $T^{\pm}$.

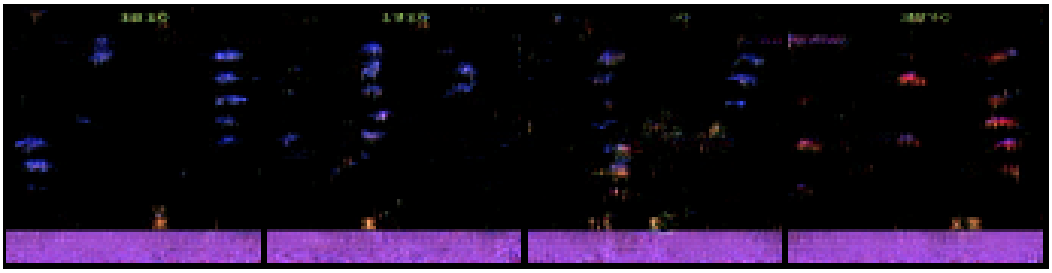


Figure 41: **Phoenix.** Target function: $T^{\pm}$.

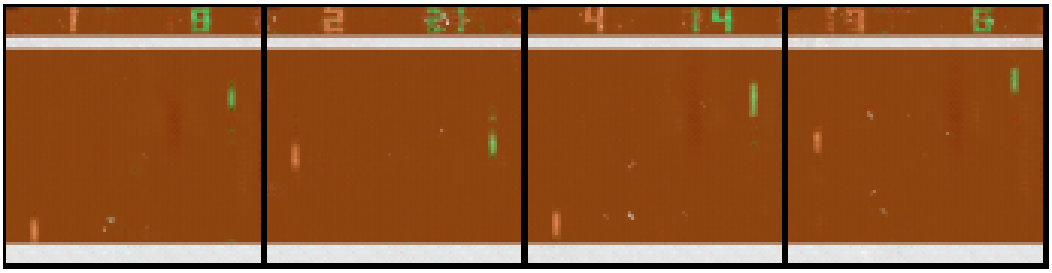


Figure 42: **Pong.** Target function: no-op.

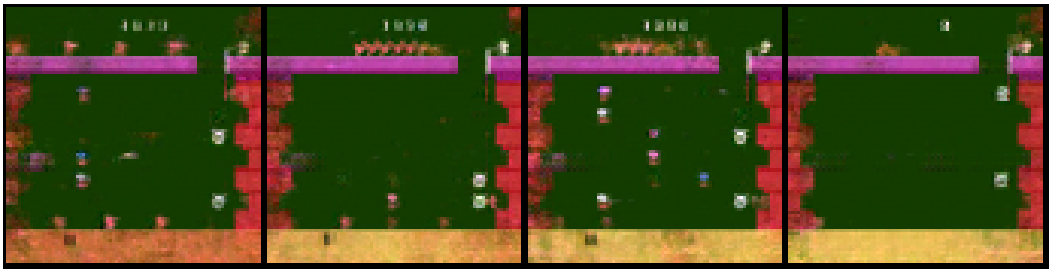


Figure 43: **Pooyan.** Target function: S^{-}.

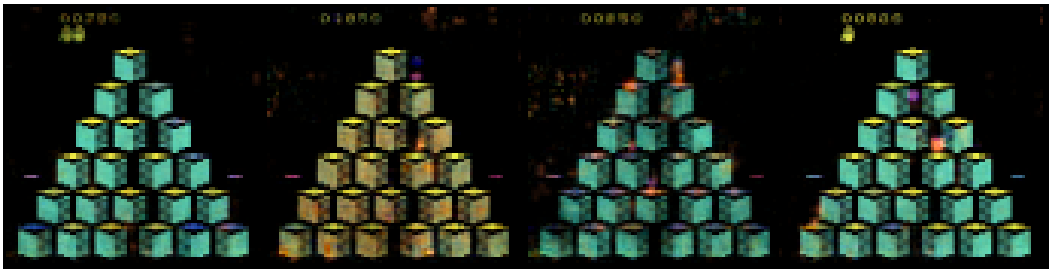


Figure 44: **Q-Bert.** Target function: left.

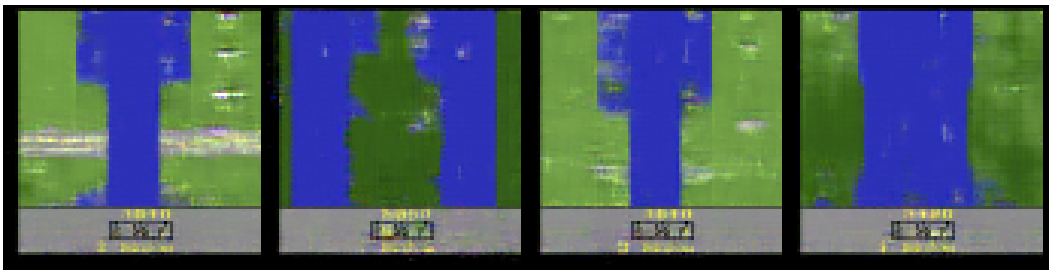


Figure 45: **River Raid.** Target function: T^{+}.

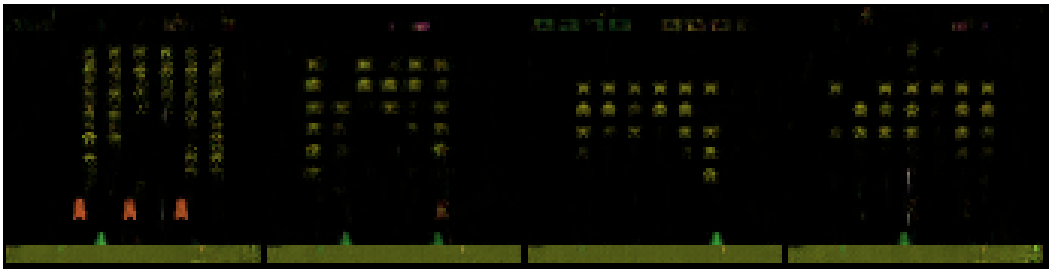


Figure 46: **Space Invaders.** Target function: left.

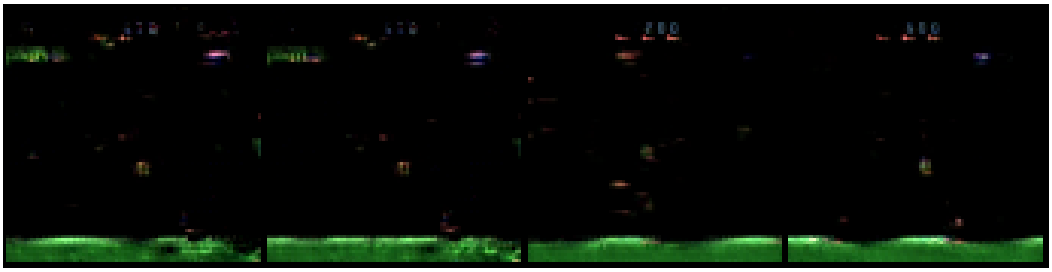


Figure 47: **Star Gunner.** Target function: $T^{\pm}$.

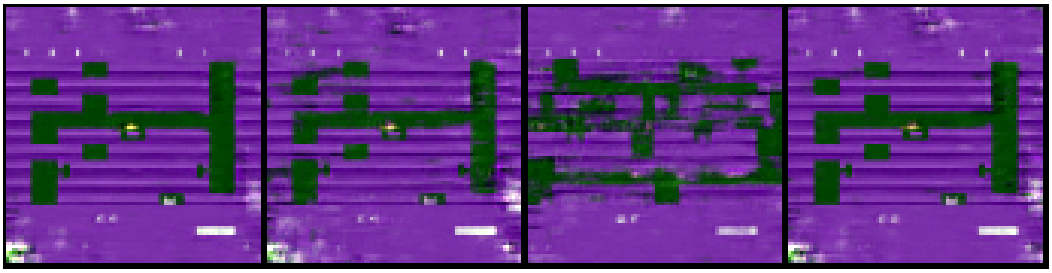


Figure 48: **Tutankham.** Target function: no-op.

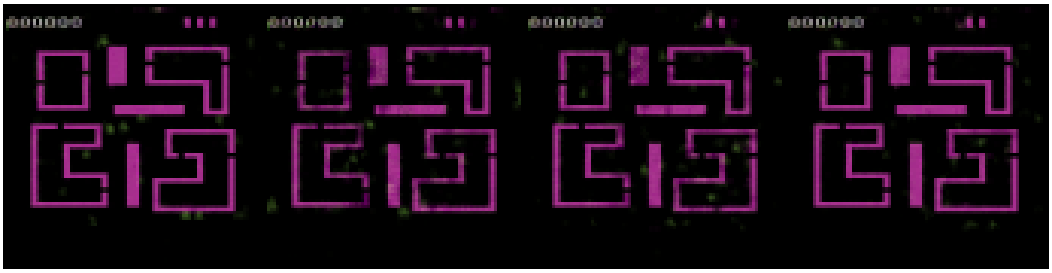


Figure 49: **Venture.** Target function: S^{+}.

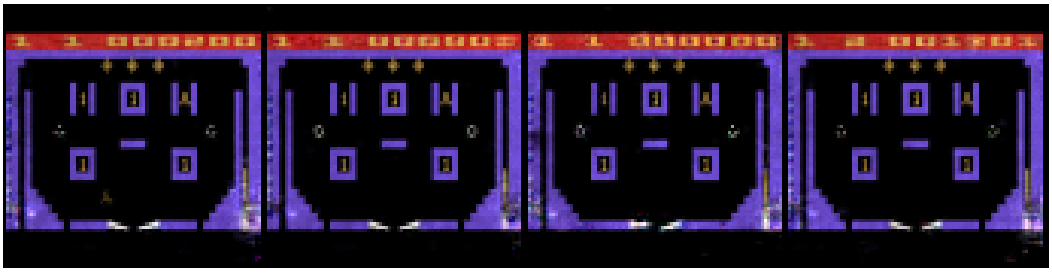


Figure 50: **Video Pinball.** Target function: T^{-}.

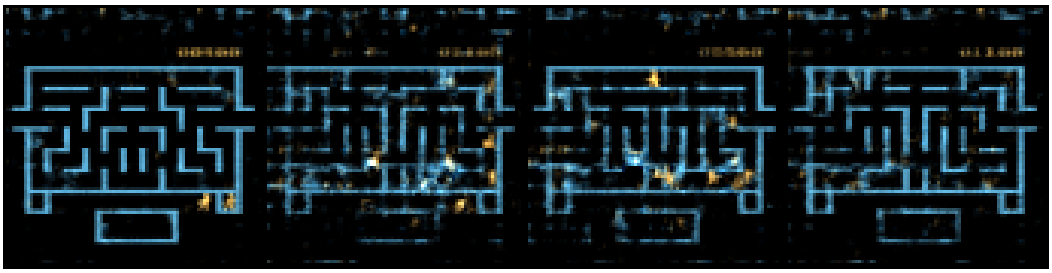


Figure 51: **Wizard Of Wor.** Target function: left.